A Generic High-performance GPU-based Library for PDE solvers

Glimberg, Stefan Lemvig; Engsig-Karup, Allan Peter

Publication date:
2012

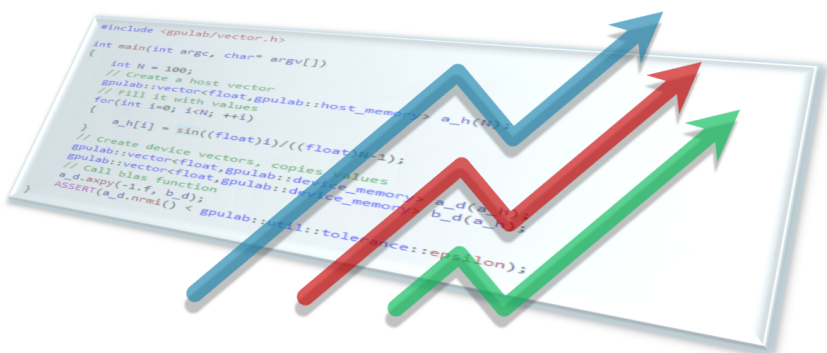
Document Version
Publisher's PDF, also known as Version of record

A Generic High-performance GPU-based Library for PDE solvers

Stefan L. Glimberg
Allan P. Engsig-Karup

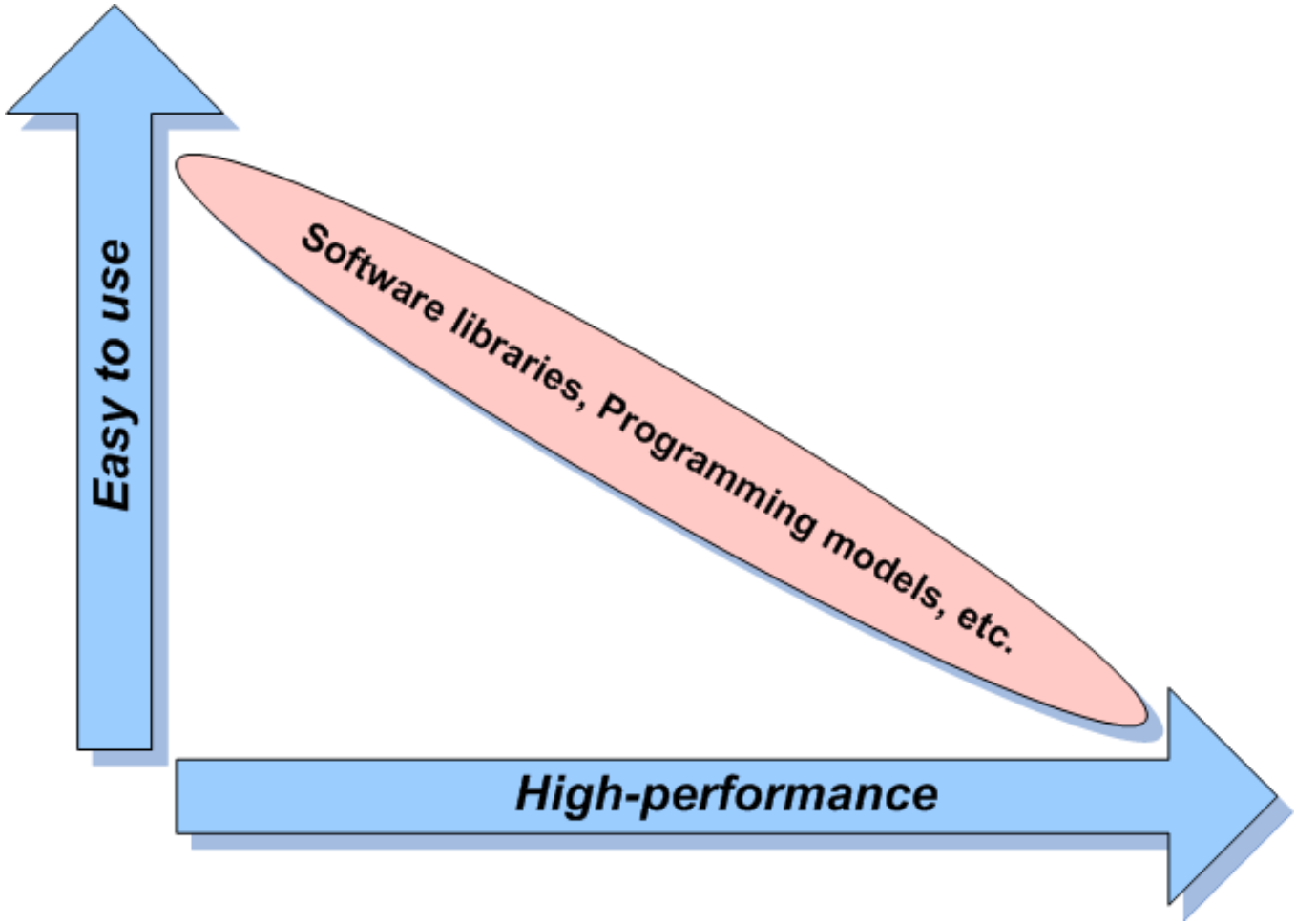
Technical University of Denmark
Department of Informatics and Mathematical Modelling
Section of Scientific Computing

PDEsoft2012
June 18^{th}, 2012



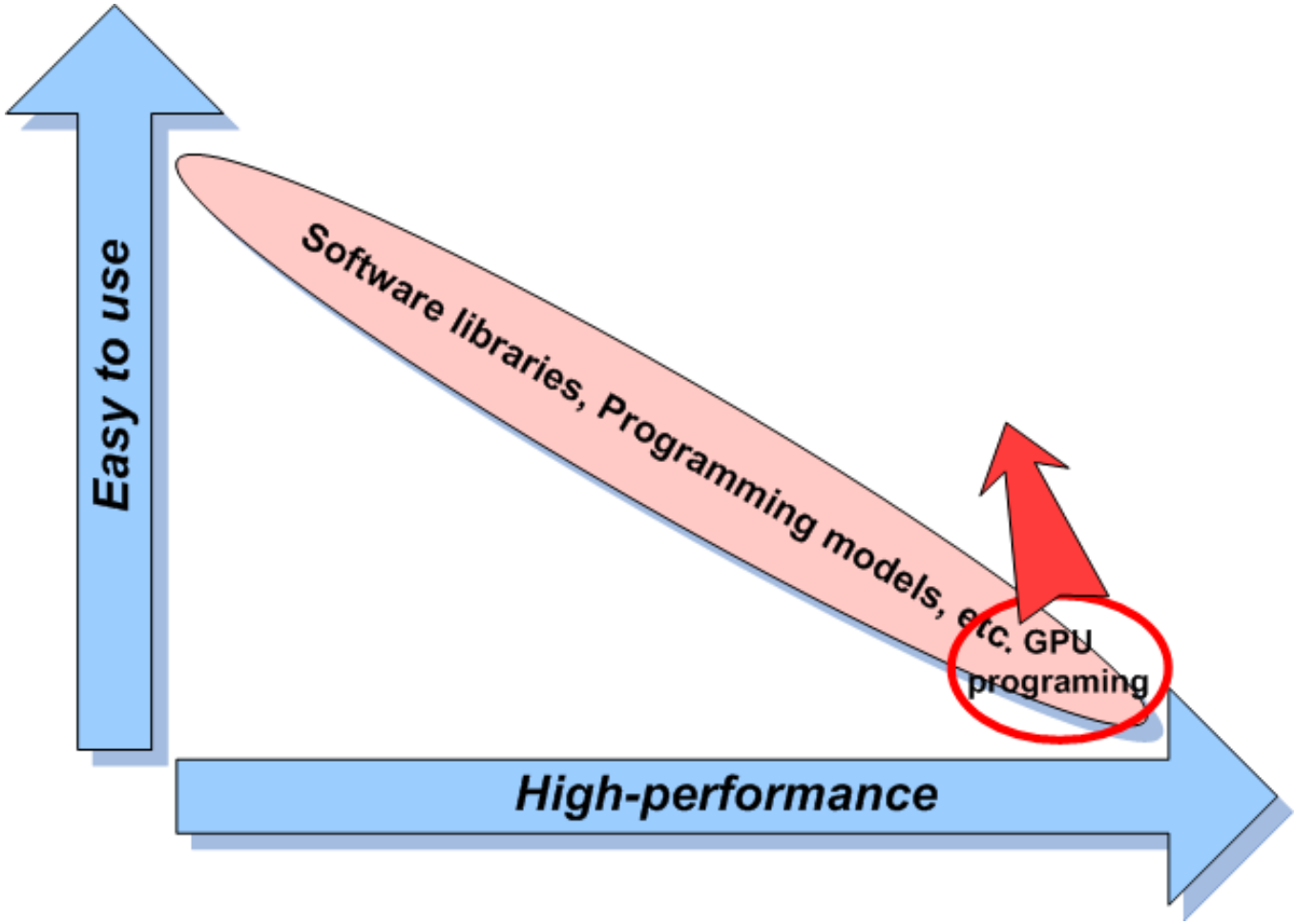
GPUlab
DTU Informatics

Tradeoff: High-level abstraction - or - Performance



Programming Models and Software Libraries

Tradeoff: High-level abstraction - or - Performance



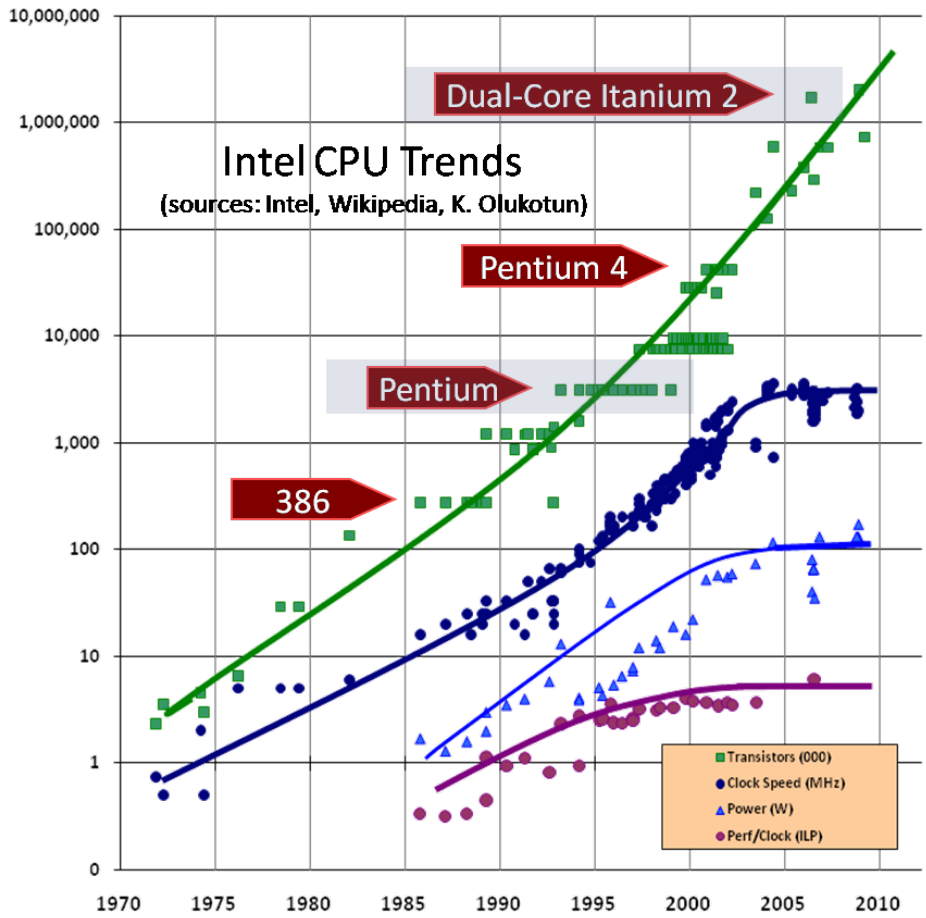
Programming GPUs efficiently is difficult and time consuming. Let software libraries help.

The Many-Core Era is here

For the last 10 years, increased performance has gone from increased clock frequency to increased core count.

Major challenges:

- Rewrite sequential code (The free lunch is over)
- Expose parallelism
- Deal with communication patterns
- Identify the right hardware for a given problem
- Performance-portable algorithms

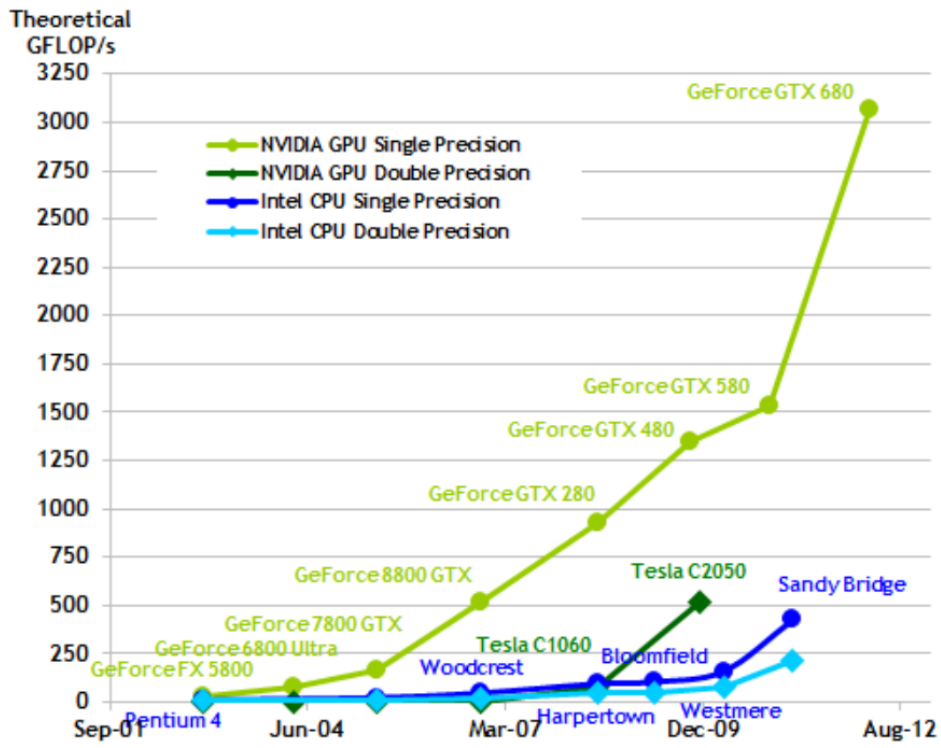


GPUlab
DTU Informatics

Motivation for CPU/GPU Heterogeneous Computing

We believe that Graphical Processing Units (GPU) are the forerunners for future high-performance many-core accelerators for scientific computing applications. Reasons to consider GPUs for high-performance computing:

- Massively parallel architecture ($1,536$ cores in Kepler GK104)
- Moderate prices $\$100 - \$2,000$, affordable personal super computer
- Energy/Cost efficient (perf/Watt)
- Tflops theoretical performance
- High memory bandwidth (Limit for many problems)



Notice this is theoretical performance.

GPUlab DTU Informatics

A GPU-based Framework for PDE Solvers

We have build a highly generic C++ heterogenous CPU/GPU framework for fast PDE solver prototyping.

Framework Objectives

- Hide GPU-specific code for basic and frequent operations to get quickly started
- While maintaining the possibility to customize code at kernel level

```
1  gpulab::vector<float,host_memory>    x_h(100,3.f); // Create host vector x, size 100, value 3
2  gpulab::vector<float,device_memory>  x_d(x_h);     // Create device vector x, transfer host data
3  gpulab::vector<float,device_memory>  y_d(x_d);     // Create device vector y, copy device data
4  y_d.axpy(4.f,x_d);                                 // Do y = a*x+y on the device
5  float n = y_d.nrm2();                              // Calculate the 2-norm on the device
```

Easy to get started and easy to use. You can still access device pointers and use them in your custom implementations.

GPUlab DTU Informatics

A GPU-based Framework for PDE Solvers

We have build a highly generic C++ heterogenous CPU/GPU framework for fast PDE solver prototyping.

Framework Objectives

- Hide GPU-specific code for basic and frequent operations to get quickly started
- While maintaining the possibility to customize code at kernel level

```
1  gpulab::vector<float,host_memory>    x_h(100,3.f); // Create host vector x, size 100, value 3
2  gpulab::vector<float,device_memory>  x_d(x_h);     // Create device vector x, transfer host data
3  gpulab::vector<float,device_memory>  y_d(x_d);     // Create device vector y, copy device data
4  y_d.axpy(4.f,x_d);                                 // Do y = a*x+y on the device
5  float n = y_d.nrm2();                              // Calculate the 2-norm on the device
```

Easy to get started and easy to use. You can still access device pointers and use them in your custom implementations.

GPUlab DTU Informatics

Library Components

The library contains several useful components for assembling PDE solvers.

Components

- Vectors (CPU / GPU)
- Regular grids, 1D, 2D, 3D.
- Compact flexible order finite difference operators (matrix free)
- Iterative solvers for solving linear system of equations, CG, GMRES, Defect Correction, Multigrid.
- Preconditioning strategies
- I/O operations

Template-based implementations enable users to use their own user-specific implementations as long as they follow some simple rules (concepts).

Library Components

The library contains several useful components for assembling PDE solvers.

Components

- Vectors (CPU / GPU)
- Regular grids, 1D, 2D, 3D.
- Compact flexible order finite difference operators (matrix free)
- Iterative solvers for solving linear system of equations, CG, GMRES, Defect Correction, Multigrid.
- Preconditioning strategies
- I/O operations

Template-based implementations enable users to use their own user-specific implementations as long as they follow some simple rules (concepts).

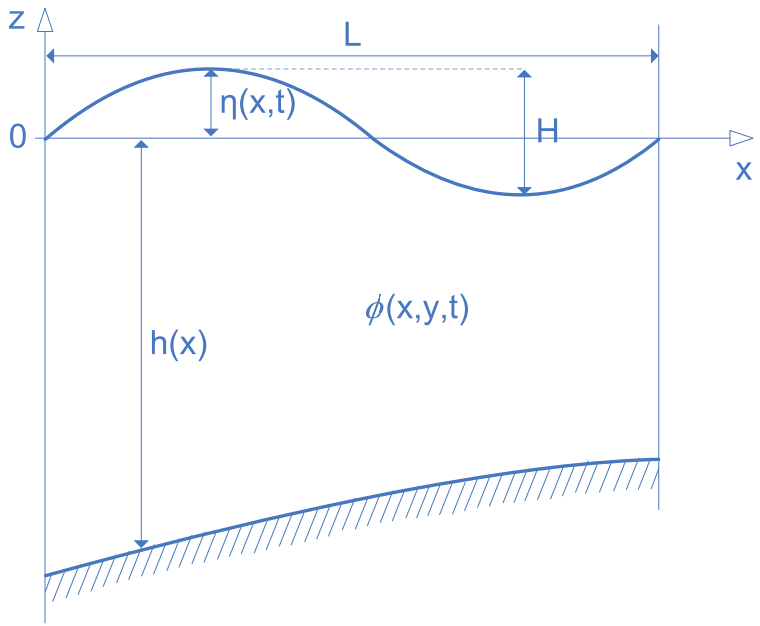
```
1  // Defect Correction setup
2  typedef gpulab::solvers::defect_correction_types<
3    vector_type
4  , matrix_type
5  , preconditioner_type> dc_types;
6  typedef gpulab::solvers::defect_correction<dc_types>  dc_solver_type;
7
8  // In main
9  dc_solver_type solver(A);
10 solver.solve(x,b);
```

Jlab matics

Fully Nonlinear Free Surface Water Waves

The potential flow equations describe fully nonlinear water waves under the assumption of incompressible, inviscid and irrotational flow with non-breaking waves.

2D Potential Flow Equations



Kinematic and dynamic free surface conditions

$$\partial_t \eta = -\partial_x \eta \, \partial_x \tilde{\phi} + \tilde{w}(1 + (\partial_x \eta)^2)$$

$$\partial_t \tilde{\phi} = -g\eta - \frac{1}{2}((\partial_x \tilde{\phi})^2 - \tilde{w}^2(1 + (\partial_x \eta)^2))$$

$$\tilde{w} = \partial_z \tilde{\phi}, \quad \tilde{\phi} = \phi|_{z=\eta}$$

For $\tilde{w}$ to be computed, we need to know the potential in the entire domain.

$$\begin{aligned} \phi &= \tilde{\phi}, && z = \eta \\ \partial_{xx}\phi + \partial_{zz}\phi &= 0, && -h \le z < \eta \\ \partial_z \phi + \partial_x h \, \partial_x \phi &= 0, && z = -h \end{aligned}$$

GPUlab DTU Informatics

Fully Nonlinear Free Surface Water Waves

The potential flow equations describe fully nonlinear water waves under the assumption of incompressible, inviscid and irrotational flow with non-breaking waves.

2D Potential Flow Equations

The potential flow model is an attractive prototype problem, because it illustrates both the capability and performance of our library well.

- Hyperbolic surface equations
- Computational intense Laplace problem
- Benefits from parallel optimizations
- Used in actual coastal engineering applications

We have shown that an iterative mixed-precision Defect Correction method with multigrid preconditioning solves the Laplace problem very effectively [Engsig-KarupEtAl2011] and [GlimbergEtAl2011].

GPUlab DTU Informatics

Assembling the Laplace Solver

The generic nature of the library allows solvers to be assembled from standard components.

```
1 // Basics
2 typedef gpulab::device_memory                                        memory_space;
3 typedef float                                                        value_type;
4 typedef gpulab::grid<value_type,memory_space>                        vector_type;
5 typedef gpulab::free_surface::laplace_sigma_stencil_3d<vector_type> matrix_type;
```

Assembling the Laplace Solver

The generic nature of the library allows solvers to be assembled from standard components.

```
1  // Basics
2  typedef gpulab::device_memory                                          memory_space;
3  typedef float                                                          value_type;
4  typedef gpulab::grid<value_type,memory_space>                          vector_type;
5  typedef gpulab::free_surface::laplace_sigma_stencil_3d<vector_type>    matrix_type;
6
7  // Multigrid setup
8  typedef gpulab::solvers::multigrid_types<
9       vector_type
10     ,matrix_type
11     ,gpulab::free_surface::jacobi_low_order_3d
12     ,gpulab::solvers::grid_handler_3d_boundary>                        multigrid_types;
13 typedef gpulab::solvers::multigrid<multigrid_types>                    preconditioner_type;
```

Assembling the Laplace Solver

The generic nature of the library allows solvers to be assembled from standard components.

```
1  // Basics
2  typedef gpulab::device_memory                                        memory_space;
3  typedef float                                                        value_type;
4  typedef gpulab::grid<value_type,memory_space>                        vector_type;
5  typedef gpulab::free_surface::laplace_sigma_stencil_3d<vector_type> matrix_type;
6
7  // Multigrid setup
8  typedef gpulab::solvers::multigrid_types<
9       vector_type
10     ,matrix_type
11     ,gpulab::free_surface::jacobi_low_order_3d
12     ,gpulab::solvers::grid_handler_3d_boundary>                      multigrid_types;
13 typedef gpulab::solvers::multigrid<multigrid_types>                  preconditioner_type;
14
15 // DC setup
16 typedef gpulab::solvers::defect_correction_types<
17      vector_type
18     ,matrix_type
19     ,preconditioner_type>                                            dc_types;
20 typedef gpulab::solvers::defect_correction<dc_types>                 laplace_solver_type;
```

Assembling the Laplace Solver

The generic nature of the library allows solvers to be assembled from standard components.

```
1  // Basics
2  typedef gpulab::device_memory                                          memory_space;
3  typedef float                                                          value_type;
4  typedef gpulab::grid<value_type,memory_space>                          vector_type;
5  typedef gpulab::free_surface::laplace_sigma_stencil_3d<vector_type>    matrix_type;
6
7  // Multigrid setup
8  typedef gpulab::solvers::multigrid_types<
9       vector_type
10     ,matrix_type
11     ,gpulab::free_surface::jacobi_low_order_3d
12     ,gpulab::solvers::grid_handler_3d_boundary>                        multigrid_types;
13 typedef gpulab::solvers::multigrid<multigrid_types>                    preconditioner_type;
14
15 // DC setup
16 typedef gpulab::solvers::defect_correction_types<
17      vector_type
18     ,matrix_type
19     ,preconditioner_type>                                              dc_types;
20 typedef gpulab::solvers::defect_correction<dc_types>                   laplace_solver_type;
```

```
1  // Somewhere in main
2  laplace_solver_type solver(A);     // Create solver
3  solver.solve(x,b);                 // Solve Ax=b
```

GPUlab DTU Informatics

Adding Mixed-Precision

A mixed precision algorithm performs part of its calculations in single (low) precision arithmetics, while maintaining the solution in double (high) precision. Refinement processes are applicable for this technique (preconditioning in the Defect Correction method).

Adding Mixed-Precision

A mixed precision algorithm performs part of its calculations in single (low) precision arithmetics, while maintaining the solution in double (high) precision. Refinement processes are applicable for this technique (preconditioning in the Defect Correction method).

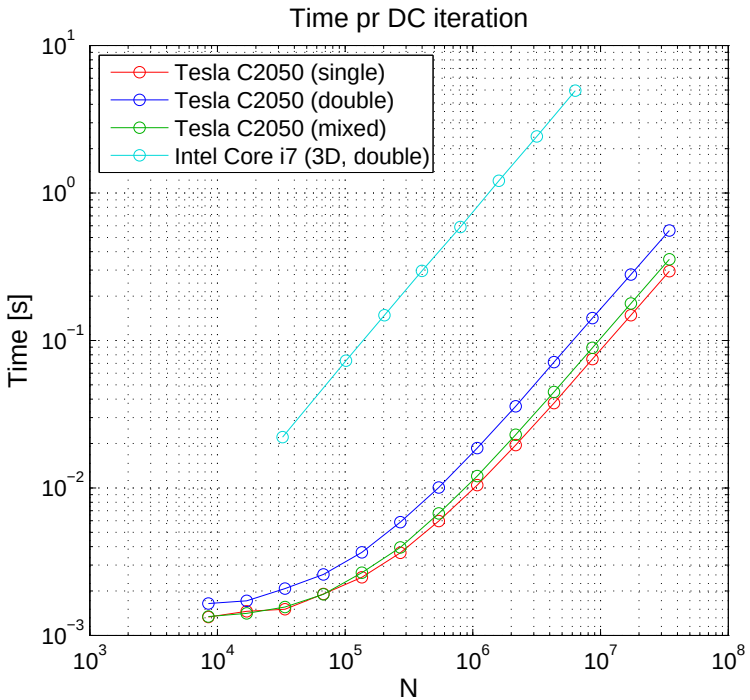
A few lines of code does the trick

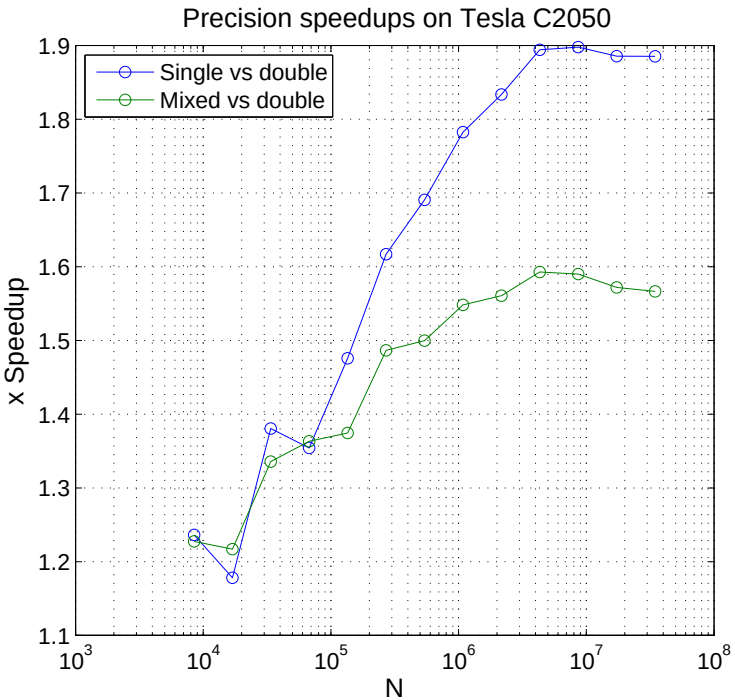
```
1  // Basics
2  typedef gpulab::device_memory                                                memory_space;
3  typedef double                                                               value_type;
4  typedef float                                                                value_type_low;
5
6  typedef gpulab::grid<value_type,memory_space>                                vector_type;
7  typedef gpulab::grid<value_type_low,memory_space>                            vector_type_low;
8
9  typedef gpulab::free_surface::laplace_sigma_stencil_3d<vector_type>     matrix_type;
10 typedef gpulab::free_surface::laplace_sigma_stencil_3d<vector_type_low> matrix_type_low;
11
12 // Create multigrid with vector_type_low and matrix_type_low
13 ...
14
15 // Create defect correction with vector_type and matrix_type
16 ...
```

GPUlab DTU Informatics

2D Performance Results

Proof-of-concept has first been established in 2D.



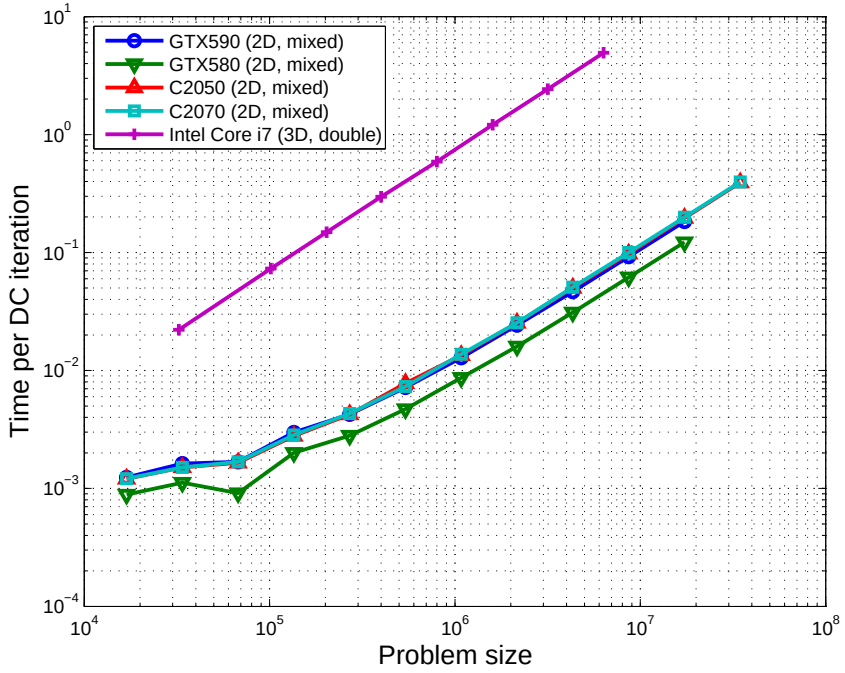


Timings per Defect Correction iteration. Using 6^{th} order accurate stencil, preconditioned with a linear 2^{nd} order accurate multigrid method, DC+MG-RB-GS-V(2,2).

GPUlab DTU Informatics

2D Performance Results

Proof-of-concept has first been established in 2D.

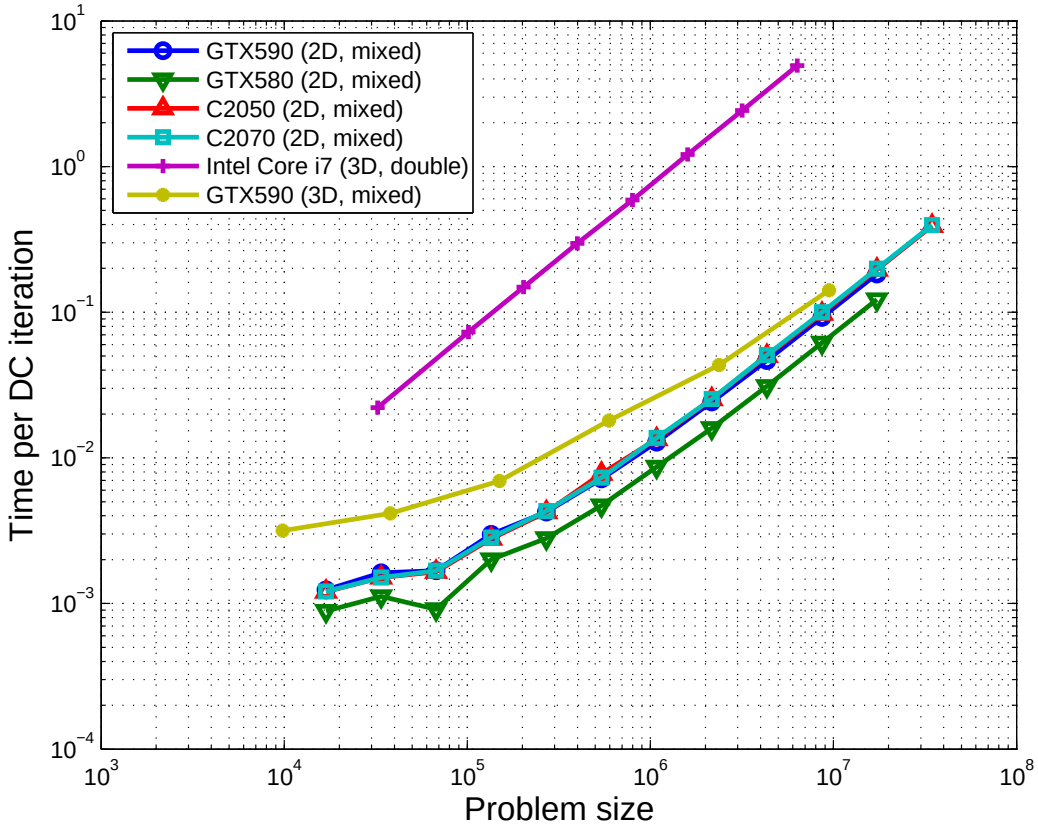


Timings per Defect Correction iteration. Using 6^{th} order accurate stencil, preconditioned with a linear 2^{nd} order accurate multigrid method, DC+MG-RB-GS-V(2,2).

GPUlab DTU Informatics

3D Performance Results

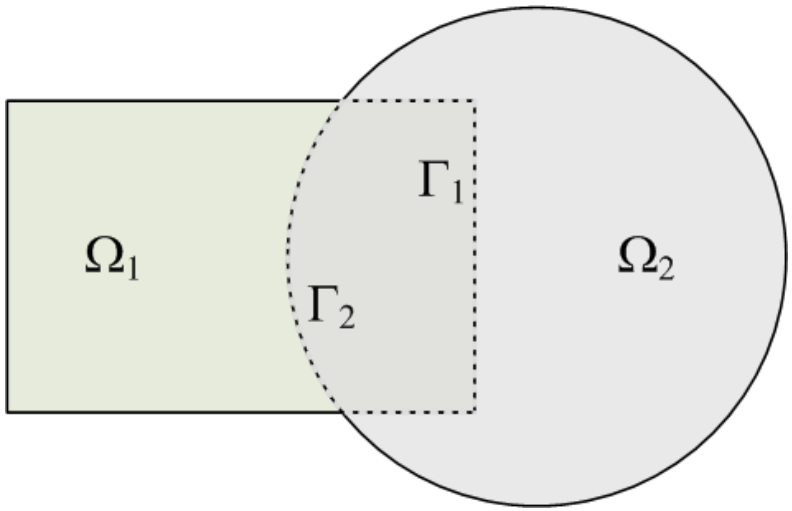
We are now working on a 3D version of the same solver. Recent results are not optimized, yet they show promising behavior.



GPUlab DTU Informatics

Domain Decomposition

In order to increase performance and/or resolution, we split the problem into multiple domains to be solved on multiple GPUs. MPI is used to communicate between nodes.



Solve $\mathcal{L}u = f$ on $\Omega = \Omega_1 \bigcup \Omega_2$ with boundary conditions $u = g$ on $\partial\Omega$.

Schwartz Method

First solve

$$\mathcal{L}u_1^{(k+1)} = f$$
$$u_1^{(k+1)} = g \quad \text{on } \partial\Omega \backslash \Gamma_1$$
$$u_1^{(k+1)} = u_2^{(k)} \quad \text{on } \Gamma_1$$

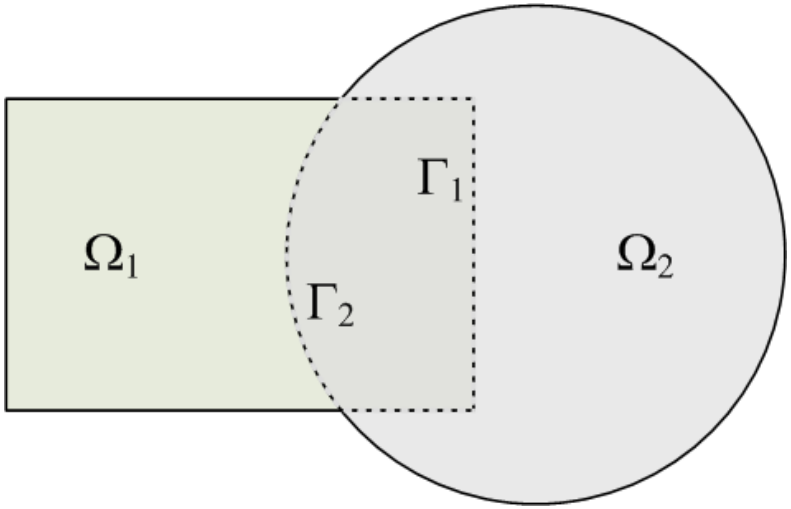
Then solve

$$\mathcal{L}u_2^{(k+1)} = f \tag{1}$$
$$u_2^{(k+1)} = g \quad \text{on } \partial\Omega \backslash \Gamma_2 \tag{2}$$
$$u_2^{(k+1)} = u_1^{(k+1)} \quad \text{on } \Gamma_2 \tag{3}$$

Until convergence.

Domain Decomposition

In order to increase performance and/or resolution, we split the problem into multiple domains to be solved on multiple GPUs. MPI is used to communicate between nodes.



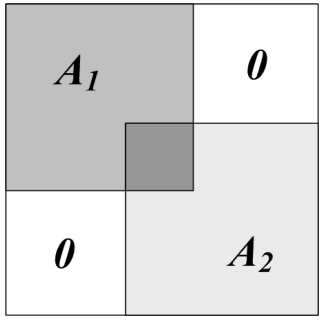
Solve $\mathcal{L}u = f$ on $\Omega = \Omega_1 \bigcup \Omega_2$ with boundary conditions $u = g$ on $\partial\Omega$.

Schwartz Method

In discrete form let $\mathbf{R_i}$ be the $n_i \times n$ matrix that selects $\mathbf{A_i}$ corresponding to Ω_i from $\mathbf{A}$.

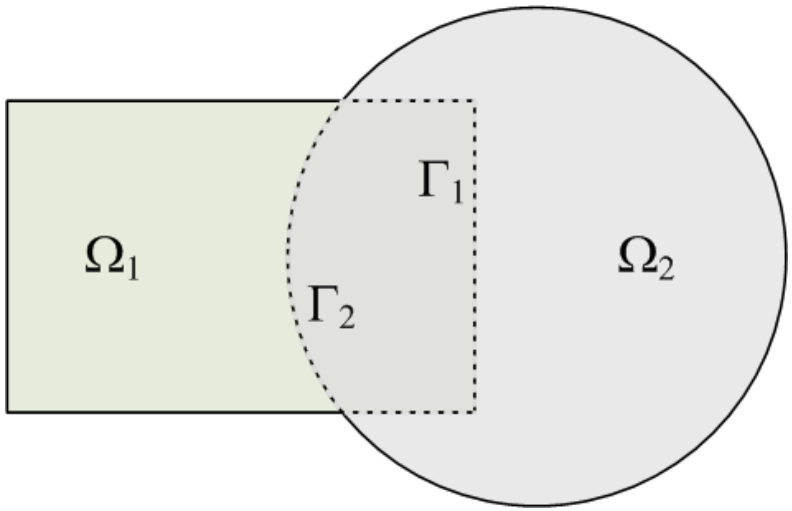
$$\mathbf{A_1} = \mathbf{R_1 A R_1^T}$$

$$\mathbf{A_2} = \mathbf{R_2 A R_2^T}$$



Domain Decomposition

In order to increase performance and/or resolution, we split the problem into multiple domains to be solved on multiple GPUs. MPI is used to communicate between nodes.



Solve $\mathcal{L}u = f$ on $\Omega = \Omega_1 \bigcup \Omega_2$ with boundary conditions $u = g$ on $\partial\Omega$.

Schwartz Method

The additive Schwartz method:

$$\mathbf{x}_{k+1} = \mathbf{x}_{k+\frac{1}{2}} + (\mathbf{R}_1^T \mathbf{A}_1^{-1} \mathbf{R}_1 + \mathbf{R}_2^T \mathbf{A}_2^{-1} \mathbf{R}_2)(\mathbf{b} - \mathbf{A}\mathbf{x}_k)$$

Which is a defect correction iterations with an additive Schwartz preconditioner

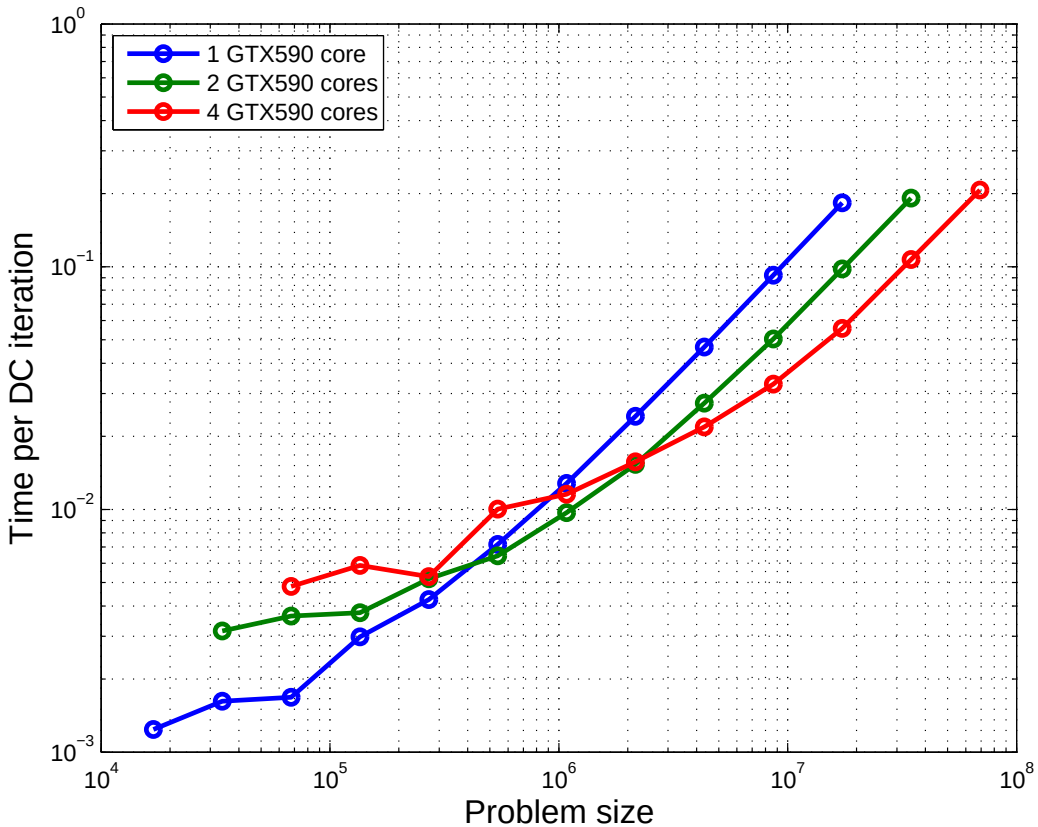
$$\mathbf{M} = \sum_{i=1}^{p} \mathbf{R}_i^T \mathbf{A}_i^{-1} \mathbf{R}_i,$$

where $\mathbf{M}$ is the preconditioner.

GPUlab DTU Informatics

2D Domain Decomposition Performance Results

For large scale problems communication time becomes minor and the overall performance seems promising.



Same problem setup as previous performance figures.

GPUlab DTU Informatics

Future Work and Ideas

- Add 3D domain decomposition and demonstrate scalability behavior at very large scales (strong and weak scaling).
- Extend library with support for multi GPU computations (MPI helpers)
- Autotuning to optimize library for various GPU architectures.
- Parareal, can we do efficiently parallel time integration on the GPU?
- Proof-of-concept prototype for real-time simulation of ship-wave and ship-ship interactions (tool to strengthen naval educations).

Bibliography

Allan Peter Engsig-Karup, Morten Gorm Madsen, and Stefan Lemvig Glimberg.

A massively parallel gpu-accelerated model for analysis of fully nonlinear free surface waves.

International Journal for Numerical Methods in Fluids, 2011.

Stefan Lemvig Glimberg, Allan Peter Engsig-Karup, and Morten Gorm Madsen.

A fast gpu-accelerated mixed-precision strategy for fully nonlinear water wave computations.

Proceedings of European Numerical Mathematics and Advanced Applications (ENUMATH), 2011.

Figures from http://www.gotw.ca/publications/concurrency-ddj.htm and www.nvidia.com